Planning in Stochastic Domains: Problem Characteristics and Approximations (Version II)

*Nevin L. Zhang and Wenju Liu**

Technical Report HKUST-CS96-31

*Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Abstract

This paper is about planning in stochastic domains by means of partially observable Markov decision processes (POMDPs). POMDPs are difficult to solve and approximation is a must in real-world applications. Approximation methods can be classified into those that solve a POMDP directly and those that approximate a POMDP model by a simpler model. Only one previous method falls into the second category. It approximates POMDPs by using (fully observable) Markov decision processes (MDPs). We propose to approximate POMDPs by using what we call region observable POMDPs. Region observable POMDPs are more complex than MDPs and yet still solvable. They have been empirically shown to yield significantly better approximate policies than MDPs.

In the process of designing an algorithm for solving region observable POMDPs, we also propose a new method for attacking the core problem, known as dynamic-programming updates, that one has to face in solving POMDPs. We have shown elsewhere that the new method is significantly more efficient that the best previous method.

Keywords: planning under uncertainty, partially observable Markov decision processes, problem characteristics, policy trees, parsimonious coverings.

1 Introduction

To plan is to find a policy that will lead an agent to achieve a goal in the fewest number of steps possible. When the environment of the agent, henceforth referred to as the world, is completely observable and the effects of actions are deterministic, planning is reduced to finding the shortest sequence of actions that leads the agent to the goal.

In real-world applications, however, the world is rarely completely observable and effects of actions are almost always nondeterministic. For this reason, a growing number of researchers concern themselves with planning in stochastic domains (e.g. Dean and Wellman 1991, Cassandra *et al* 1994, Parr and Russell 1995). Partially observable Markov decision processes (POMDPs) can be used as a model for planning in such domains. In this model, nondeterminism in effects of actions is encoded by transition probabilities, partial observability of the world by observation probabilities, and goals and criteria for good plans by reward functions (see Section 2 for details).

POMDPs are classified into *finite horizon* POMDPs and *infinite horizon* POMDPs depending on the number of time points considered. Infinite horizon POMDPs are usually used for planning since one typically does not know beforehand the number of steps it takes to achieve a goal. This paper is concerned with how to solve an infinite horizon POMDP.

1.1 Previous work

It is well known that infinite horizon POMDPs can be approximated by finite horizon POMDPs to arbitrary precision (Section 3). Discussions in rest of this introduction is restricted to finite horizon POMDPs.

When the world is fully observable, a POMDP reduces to a *Markov decision process* (MDP). MDPs have been studied extensively in the dynamic-programming literature (e.g. Puterman 1990, Bertsekas 1987, White 1993). Recent works have been concentrated on how to deal with applications where the world can be in a large number of states (Dean *et al* 1993, Boutillier *et al* 1995).

We are concerned with the partially observable case. This case is considerably more difficult than the fully observable case for two related reasons. Firstly, when the agent knows exactly which state the world is currently in, information from the past — past observations and actions — is irrelevant to the current decision. This is the Markov property. On the other hand, when the agent does not fully observe the state of the world, past information becomes relevant because it can help the agent to better estimate the true current state of the world. The problem is that the number of possible states of past information increases exponentially with time.

Secondly, in MDPs the effects of an action are fully observed at the next time point. In POMDPs, on the other hand, the effects of an action are not fully observed at the next time point. Hence the effects of the action is not separable from those of the agent's future behaviors. To properly evaluate the effects of an action, one needs to consider the agent's possible future behaviors. The problem is that the number of ways the agent can behave in the future is exponential in the length of the remaining of the planning horizon.

Previous methods for solving finite horizon POMDPs are usually classified into exact

methods and approximate methods (Lovejoy 1991a). They can also be classified according to which of the above two difficulties they directly address. Most previous methods address the difficulty of exponential number of future behaviors and rely on the idea of selecting a minimal subset of future behaviors to cover the set of all future behaviors in the following sense: No matter what is known about the current true state of the world, there exists at least one behavior in the subset that is optimal (Section 4). We call this class of methods *policy-oriented methods* and they include the approaches by Sondik (1971), Sondik and Mendelssohn (1979, see Lovejoy 1991a), Monahan (1982), Cheng (1988), Lovejoy (1991b), and Cassandra *et al* (1994). Other methods such as those by Platzman (1977, see Lovejoy 1991a), and White and Schere (1994) approximates the exponential set of past information states by a subset and we call them *information-oriented methods*.

All the methods mentioned above solve POMDPs directly. A recent approach by Cassandar *et al* (1996) approximates a POMDP by using an MDP and constructs an approximate solution to the POMDP from the solution of the MDP.

All the methods that solve POMDPs directly quickly break down when the problem size becomes large. And MDPs are usually not good approximations of POMDPs.

1.2 Our idea

We make the following assumption about problem characteristics. Even though an agent who acts in a stochastic domain does not know the true state of the world, he should often has a good idea about it. When the agent get confused about the state of the world sometimes, several information gathering steps should disambiguate the uncertainty.

Philosophically, one could not expect an agent to achieve any goal if the agent was constantly lost about the state of the world and could not rectify the situation by gathering information. To justify the assumption by examples, consider robot path planning. Observing a landmark, a room number for instance, would imply that the robot is at the proximity of that landmark. Observing a feature about the world, a corridor T-junction for instance, might imply the robot is in one of several regions. Taking history into account, the robot might be able to determine a unique region for its current location. Also, an action usually moves the true state of the world to only a few "nearby" states. Thus if the robot has an good idea about the current state of world, it should continue to have a good idea about it in the next few steps.

To exploit such problem characteristics, we transform a POMDP by assuming that in addition to the observations obtained by himself, the agent also receives a report from an oracle who knows the true state of the world. The oracle does not report the true state itself. Rather he reports that the true state is a certain region. The transformed POMDP is said to be *region observable* for obvious reasons. If the agent already knows that the true state is roughly in a region and the oracle reports that region, then not much extra information is provided. In this case, the region observable POMDP should be a good approximation of the original POMDP.

When the oracle is allowed to only report singleton regions, he reports the true state of the world. In this case, the region observable POMDP is actually an MDP and hence is easy to solve. One would expect that the region observable POMDP to be solvable when the oracle is allowed to report only small regions.

Our proposal to approximate POMDPs by using region observable POMDPs is a generalization of the idea of approximating POMDPs by MDPs. It is orthogonal to all methods that solve POMDPs directly; all those methods can be used to solve region observable POMDPs.

1.3 Organization

The rest of this paper is organized as follows. We shall first briefly show how POMDPs can be used as a model for planning (Section 2) and give a new and concise account of the theory of POMDPs (Sections 3 and 4). We shall then formally introduce the concept of region observable PODMPs (Section 5). Thereafter, we shall describe an algorithm for solving region observable POMDPs (Sections 6, 7, and 8). Along the way, we shall propose a new method for solving the core problem, known as dynamic-programming updates, that one has to face in solving POMDPs (Section 7). Section 9 will discuss a couple of different ways of making decisions based on the solutions of region observable POMDPs and present a way for information gathering. Finally, empirical results will be reported in Section 10 and conclusion will be provided in Section 11.

2 Planning in stochastic domains and POMDPs

To specify a planning problem, one needs to give a set $\mathcal{S}$ of possible states of the world, a set $\mathcal{O}$ of possible observations, and a set $\mathcal{A}$ of possible actions. In this paper, all those three sets are assumed to be finite. One needs also to give an observation model, which describes the relationship between an observation and the state of the world; and an action model, which describes the effects of each action. Furthermore, one needs to specify the initial state of the world and a goal state.

As a background example, consider path planning for a robot who acts in an office environment. Here $\mathcal{S}$ is the set of all location-orientation pairs, $\mathcal{O}$ is the set of possible sensor readings, and $\mathcal{A}$ consists of actions `move-forward`, `turn-left`, `turn-right`, and `declare-goal`.

The current observation o depends on the current state of the world s. Due to sensor noise, this dependency is uncertain in nature. The observation o sometimes also depends on the action that the robot has just taken a_{-}. The minus sign in the subscript indicates the previous time point. In the POMDP model, the dependency of o upon s and a_{-} is numerically characterized by a conditional probability $P(o|s, a_{-})$, which is usually referred to as the *observation probability*. It is the observation model.

In a region observable POMDP, the current observation also depends on the previous state of the world s_{-}. The observation probability for this case can be written $P(o|s, a_{-}, s_{-})$.

The state s_{+} the world will be in after taking an action a depends on the action and on the current state s. The plus sign in the subscript indicates the next time point. This dependency is again uncertain in nature due to uncertainty in the actuator. In the POMDP model, the dependency of s_{+} upon s and a is numerically characterized by a conditional probability $P(s_{+}|s, a)$, which is usually referred to as the *transition probability*. It is the action model.

On many occasions, we will need to consider the joint conditional probability $P(s_+, o_+|s, a)$ of the next state of the world and the next observation given the current state and the current action. It is given by

$$P(s_+, o_+|s, a) = P(s_+|s, a)P(o_+|s_+, a, s_-).$$

The POMDP model encodes the starting state by a probability mass function P_0 over $\mathcal{S}$. The planning goal is encoded by a *reward function* such as the following:

$$r(s, a) = \begin{cases} 1 & \text{if } a\text{=}\texttt{delcare-goal} \text{ and } s\text{=}\texttt{goal}, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

The preference for short plans is encoded by discounting future rewards with respect the current reward (see the next section).

3 Basics of POMDPs

3.1 Belief states

In a POMDP, an agent chooses and executes an action at each time point. The choice is to be made based on information from past — past observations and past actions — and the current observation. The amount of memory required to store past observations and actions increases linearly with time. This makes it difficult to maintain past information after a long period of time.

The standard way to overcome this difficulty is to maintain, instead of past information, the agent's *belief state* — the probability distribution of the current state of the world given past information and the current observation $P(s_t|o_t, a_{t-1}, o_{t-1}, \ldots, a_1, o_1, P_0)$. It is well known that the belief state is a *sufficient statistic* in the sense that it captures all the information contained in past information and the current observation that is useful for action selection. Hence the agent can base its decision solely on the belief state.

Compared with maintaining past information, maintaining the belief state is desirable because the number of possible states of the world is finite. Consequently, one needs only to maintain a fixed and finite number of probability values.

The initial belief state is P_0. A question is how the agent should update its belief state as time goes by. Following Littman (1994), we use b to denote a belief state. For any state s, $b(s)$ is the probability that the world is in state s. The set of all possible belief states will be denoted by $\mathcal{B}$.

Suppose b is the current belief state, and a is the current action. If the observation o_+ is obtained at the next time point, then the agent should update its belief state from b to b_+, where b_+ is given by

$$b_+(s_+) = k \sum_s P(s_+, o_+|s, a)b(s), \quad (2)$$

where $k{=}1/\sum_{s,s_+} P(s_+, o_+|s, a)b(s)$ is the normalization constant. To signify the dependence of b_+ upon b, a, and o_+, we shall sometimes write it as $b_+(.|b, a, o_+)$.

3.2 POMDPs as MDPs

For any belief state b and any action a, define

$$r(b,a) = \sum_s b(s)r(s,a). \tag{3}$$

It is the expected immediate reward for taking action in belief state b.

For any belief state b, any action a, and any observation o_+, define

$$P(o_+|b,a) = \sum_{s,s_+} P(s_+,o_+|s,a)b(s). \tag{4}$$

It is the probability of observing o_+ at the next time point given that the current belief state is b and the current action is a. It can also be understood as the probability of the next belief state being $b_+(.|b,a,o_+)$.

A POMDP over world state space $\mathcal{S}$ can be viewed as an MDP over the belief state space $\mathcal{B}$, with reward function and transition probability given by equations (3) and (4) respectively.

3.3 Optimal policies

At each time point, the agent consults its belief state and chooses an action. A *policy* π prescribes an action for each possible belief state. Formally it is a mapping from $\mathcal{B}$ to $\mathcal{A}$. For each belief state b, $\pi(b)$ is the action prescribed by π for b.

Suppose b_0 is the current belief state. If an agent follows a policy π, then his current action is $\pi(b_0)$ and the immediate reward is $r_0(b,\pi(b_0))$; with probability $P(o_+|b_0,\pi(b_0))$, the agent's next belief state b_1 will be $b_+(.|b_0,\pi(b_0),o_+)$, the next action will be $\pi(b_1)$, and the next reward will be $r_1(b_1,\pi(b_1))$; and so on and so forth.

The quality of a policy is measured by the expected discounted rewards it garners. Formally the *expected discounted reward* of policy π is defined for each belief sate b_0 to be the following expectation:

$$V^\pi(b_0) = E_{b_0}[\sum_{i=0}^{\infty} \gamma^i r_i(b_i,\pi(b_i))], \tag{5}$$

where $0 \leq \gamma < 1$ is the *discount factor*.

A policy π_1 is *dominates* another policy π_2 if for each belief state $b \in \mathcal{B}$

$$V^{\pi_1}(b) \geq V^{\pi_2}(b). \tag{6}$$

Domination is a partial ordering among policies. It is well known that there exist policies that dominates all other policies. Such a policy is called an *optimal policy*. The expected discounted reward for an optimal policy will be denoted by V^*.

Consider two POMDPs that are the same except that their reward functions differ by a constant. Then the domination relationships among policies are the same in both POMDPs. For this reason, one can assume the reward function $r(s,a)$ be non-negative, which we do in the rest of this paper.

3.4 Value iteration

Value iteration is a standard way for solving MDPs. It begins with an arbitrary initial function $V_0^*(b)$ and iterates by using the following equation

$$V_t^*(b) = max_a[r(b,a) + \gamma \sum_{o_+} P(o_+|b,a)V_{t-1}^*(b_+)], \tag{7}$$

where b_+ is a shorthand for $b_+(.|b,a,o_+)$. If V_0^*=0, V_t^* is called the *t-step optimal value function*.

The following theorem (Puterman 1990, page 361) tells one when to stop and how to construct "good enough" policy.

Theorem 1 *Let π the policy given by*

$$\pi(b) = arg\ max_a[r(b,a) + \gamma \sum_{o_+} P(o_+|b,a)V_{t-1}^*(b_+)]. \tag{8}$$

If $max_{b\in\mathcal{B}}|V_t^*(b) - V_{t-1}^*(b)| \leq \epsilon$, then

$$max_{b\in\mathcal{B}}|V^\pi(b) - V^*(b)| \leq \frac{2\epsilon\gamma}{1-\gamma}. \tag{9}$$

The quantity $max_{b\in\mathcal{B}}|V_t^*(b) - V_{t-1}^*(b)|$ is sometimes called the *Bellman residual* and the policy π is called the *greedy policy* based on V_{t-1}^*.

4 Piecewise linearity and implicit value iteration

Since there are uncountably infinite many belief states, value iteration cannot to carried out explicitly. Fortunately, it can be carried out implicitly due to the piecewise linearity of the t-step optimal value function. To explain piecewise linearity, we need the concept of policy trees.

4.1 T-step policy trees

A *t-step policy tree* p_t (Littman 1994) prescribes an action for the current time point and an action for each possible information scenario $(o_1, \ldots, o_i, a_0, \ldots, a_{i-1})$ at each of the next $t-1$ time points i. Figure 1 shows 3-step policy tree. The tree reads as follows. `Move-forward` at the current time point. At the next time point, if o_1=0 is observed then `turn-left`. Thereafter if o_2=0 is observed then `turn-left` again; else if o_2=1 is observed then `declar-goal`; else if o_2=2 is observed then `move-forward`. And so on and so forth. To relate back to the introduction, a t-step policy tree prescribes a way the agent might behave at the current and the next $t-1$ time points.

When t>1, the subtree rooted at the o_1 node will be called a *o-rooted $t-1$-step policy tree*, and will be denoted by δ^{t-1}. It is a mapping from $\mathcal{O}$ to the set of all possible $t-1$-step policy trees and it prescribes a $t-1$ step policy tree $\delta^{t-1}(o)$ for each possible observation o. In our example, $\delta^1(o_1{=}0)$ is the 2-step policy tree rooted at the upper most a_1 node.

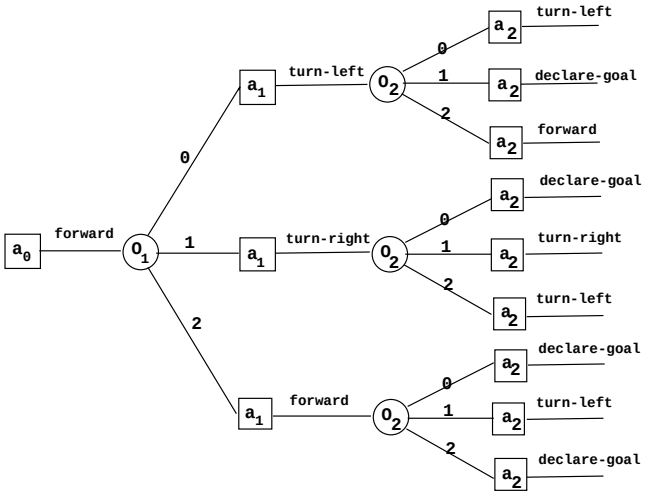

Figure 1: A 3-step policy tree.

So a t-step policy tree p_t consists of two components when $t>1$: an action a for the current time point and an o-rooted $t-1$-step policy tree δ^{t-1} for the next $t-1$ time points. For this reason, we shall sometimes write p_t as a pair (a, δ^{t-1}) and call a the *first action* of p_t.

By altering the actions on the edges out of the a-nodes, one obtains different t-step policy trees. The set of all possible t-step policy trees will be denoted by $\mathcal{P}_t$. A 1-step policy tree is simply an action, and hence $\mathcal{P}_1$ is same as the set of possible actions $\mathcal{A}$.

4.2 State value functions of t-step policy trees

For any state s and any t-step policy tree $p_t=(a, \delta^{t-1})$, recursively define

$$V_{p_t}(s) = r(s,a) + \gamma \sum_{o_+, s_+} V_{\delta^{t-1}(o_+)}(s_+) P(s_+, o_+ | s, a), \tag{10}$$

where the second term is to be understood as 0 when $t=1$. Called the *state value function* of the t-step policy tree p_t, $V_{p_t}(s)$ is the expected discounted total rewards the agent receives at the current time and during the next $t-1$ time points if the world is currently in state s and the agent behaves according to the policy tree p_t.

Without mentioning the policy tree, we shall sometimes call V_{p_t} a *t-step state value function*. The collection of all t-step state value functions will be denoted by $\mathcal{V}_t$, i.e.

$$\mathcal{V}_t = \{V_{p_t} | p_t \in \mathcal{P}_t\}.$$

For convenience, we let $\mathcal{V}_0$ consist of one single function of s that is zero for all s.

4.3 S-functions and b-functions

A function of s will be called an *s-function*, while a function of b will be called a *b-function*. The t-step state value function is an s-function, while the t-step optimal value function is

a b-function.

An s-function $V(s)$ *induces* a b-function through

$$V(b) = \sum_s V(s)b(s).$$

Regarding b is a vector with one component $b(s)$ for each s, the induced b-function is *linear* in the components of b. For convenience, we simply say that $V(b)$ is *linear* in b.

A collection $\mathcal{V}$ of s-functions *induces* a b-function through

$$\mathcal{V}(b) = max_{V \in \mathcal{V}} V(b).$$

Note that we are using $\mathcal{V}$ to denote both a set of s-functions and the b-function it induces.

The induced b-function $\mathcal{V}(b)$ is *piecewise linear* in b in the sense that it is linear in b in each of the regions $\{b | V(b) \geq V'(b)$ for any other$V' \in \mathcal{V}\}$ of the belief space $\mathcal{B}$ for each $V \in \mathcal{V}$.

4.4 Piecewise linearity of the t-step optimal value function

The following theorem was first proved by Sondik (1971). It first appeared in its present form in Littman (1994).

Theorem 2 *(***Piecewise Linearity***) The t-step optimal value function V_t^* is the same as the b-function induced by the collection of all t-step state value functions $\mathcal{V}_t$. That is for any belief state b*

$$V_t^*(b) = \mathcal{V}_t(b). \Box$$

Intuitively the theorem is true for the following reasons. $V_t^*(b)$ is the reward the agent gets if he behaves optimally and for any policy tree p_t, $V_{p_t}(b)$ is the reward the agent gets if he behaves according to p_t. Since one of the policy trees must be optimal, $V_t^*(b) = max_{p_t} V_{p_t}(b) = \mathcal{V}_t(b)$.

Because of this theorem, we shall say that the collection $\mathcal{V}_t$ of state value functions is a *representation* of V_t^*.

The theorem gives us an implicit way for carrying out value iteration. Instead of iteratively and explicitly computing the t-step optimal value function V_t^* and determining the Bellman residual from V_t^* and V_{t-1}^*, one iteratively computes the set $\mathcal{V}_t$ of all t-step state value functions and determines the Bellman residual from $\mathcal{V}_t$ and $\mathcal{V}_{t-1}$. When the Bellman residual falls below a predetermined threshold ϵ, one stops and passes $\mathcal{V}_{t-1}$ to the agent. The agent keeps $\mathcal{V}_{t-1}$. When he needs to make a decision, he consults his belief state b and finds an action using equation (8) with $V_{t-1}^*(b_+)$ replaced by $\mathcal{V}_{t-1}(b_+)$.

4.5 Parsimonious representations

Implicit value iteration, as described above, is unrealistic in practice because the size of $\mathcal{V}_t$ increases exponentially with t. As a matter of fact, the total number of t-step policy trees is

$$|\mathcal{P}_t| = |\mathcal{A}|^{\frac{|\mathcal{O}|^t - 1}{|\mathcal{O}|-1}}.$$

There are potentially the same number of t-step state value functions.

Fortunately, many of the state value functions in $\mathcal{V}_t$ can be pruned without affecting the induced b-function. Most algorithms for solving POMDPs exploit this property. Let us make the property more explicit.

A set $\mathcal{V}'$ of s-functions is a *covering* of another set $\mathcal{V}$ of s-functions if it induces the same b-function as $\mathcal{V}$ does. A *parsimonious covering* of $\mathcal{V}$ is a covering of $\mathcal{V}$ such that none of its proper subsets are coverings of $\mathcal{V}$. We shall use $\mathcal{V}^*$ to denote a parsimonious covering of $\mathcal{V}$.

Theorem 3 *All parsimonious coverings of a set of s-functions consist of the same number of s-functions.*

Proof of this theorem can be found in Appendix A. Because of this theorem, one can also define a parsimonious covering as a covering that contains the minimum number of s-functions.

Suppose the Bellman residual falls below ϵ at iteration t. Let $\mathcal{V}^*_{t-1}$ be a parsimonious covering of $\mathcal{V}_{t-1}$. It represents the $t-1$-step optimal value function V^*_{t-1} in the sense that for any belief state b, $\mathcal{V}^*_{t-1}(b)=V^*_{t-1}(b)$. It is a *parsimonious representation* because it consists of the fewest number of s-functions possible. It is the *solution* to the POMDP under discussion.

4.6 Dynamic-programming updates

Dynamic-programming updates (Cassandra *et al* 1996) refer to the problem of computing a parsimonious covering of $\mathcal{V}_t$ from a parsimonious covering of $\mathcal{V}_{t-1}$. A solution to this problem, together with the fact that $\mathcal{V}_0$ is a parsimonious covering of itself, allows one to compute the parsimonious covering of $\mathcal{V}_t$ for any t, and hence leads to a parsimonious way of carrying out implicit value iteration.

Several algorithms for dynamic-programming updates have been proposed, including the enumeration and pruning algorithms by Monahan (1992), Eagle (1984), and Larke (White 1991a), the one-pass algorithm by Sondik (1971), the linear support and relaxed region algorithms by Cheng (1988), and the witness algorithm by Cassandra *et al* (1994) and Littman (1994). The witness algorithm has been proved to be the most efficient among all those algorithms (Littman *et al* 1995).

Even though the size of a parsimonious covering of $\mathcal{V}_t$ is much smaller than that of $\mathcal{V}_t$, it is still large except for small toy problems. Approximation is a must for real-world problems.

5 Region-based approximations

We classify approximation methods into two categories. *Value function approximation methods* refer to those that approximate the optimal value functions of a POMDP directly. Methods in the category include those by Sondik and Mendelssohn (1979, see Lovejoy 1991a), Cheng (1988), Lovejoy (1991b), and Parr and Russell (1995). *Model approximation methods* refer to those that approximate a POMDP model itself by an simpler model and use the solution of the simpler model as an approximate solution to the POMDP. The method by Cassandra *et al* (1996) approximates POMDPs by using MDPs and is hence a model approximation method.

This paper proposes another model approximation method which approximates POMDPs by using what we call region observable POMDPs.

5.1 The basic idea

In a POMDP $\mathcal{M}$, the agent typically does not know the true state of the world. However, he often has a good idea about it in real-world applications. Take robot path planning for example. Observing a landmark, a room number for instance, would imply that the robot is in the proximity of that landmark. Observing an environment feature, a corridor T-junction for instance, would enable the robot to conclude from a map of the environment that it is in one of a few regions. Taking history into account, the robot might be able to determine a unique region for its current location. Furthermore, if the robot has a good idea about its current location and there is not much uncertainty in the effects of actions, then it should have a pretty good idea about its location after a few actions, even when no more informative observations are obtained.

Now consider another POMDP $\mathcal{M}'$ which is the same as $\mathcal{M}$ except that in addition to the observation made by himself, the agent also receives a report from an oracle who knows the true state of the world. The oracle does not report the true state itself. Rather he reports that the true state is in a certain region — a subset of possible states of the world.

More information is available to the agent in $\mathcal{M}'$ than in $\mathcal{M}$; extra information is provided by the oracle. Since in $\mathcal{M}$ the agent already has a good idea about the true state of the world, the oracle might not provides much extra information. Consequently, $\mathcal{M}'$ could be a good approximation of $\mathcal{M}$.

In $\mathcal{M}'$, the agent knows for sure that the true state of the world is in the region reported by the oracle. For this reason, we say that it is *region observable*. The region observable POMDP $\mathcal{M}'$ can be much easier to solve than $\mathcal{M}$ when the oracle is allowed to report only small regions. For example, if the oracle is allowed to report only singleton regions, then he actually reports the true state of the world and hence $\mathcal{M}'$ is an MDP. MDPs are much easier the solve than POMDPs.

5.2 Spectrum of approximations

If the region reported by the oracle is always the set of all possible states, then no extra information is provided. Because the report that the true state of the world is one of the

possible states of the world has no information content. In this case, $\mathcal{M}'$ has the same solution as $\mathcal{M}$ and solving $\mathcal{M}'$ is equivalent to solving $\mathcal{M}$ directly. This is one extreme of the spectrum.

At the other extreme, the oracle is allowed to report only singleton regions. In other words, he always reports the true state of the world. In this case, maximum amount of extra information is provided. $\mathcal{M}'$ is actually an MDP. It might not be a good approximation of $\mathcal{M}$ but it is much easier to solve than $\mathcal{M}$.

Previous methods for solving a POMDP either solve it directly or approximate it by using a MDP. By allowing the oracle to report regions that are neither singletons nor the set of all possible states, this paper opens up the possibility of exploring the spectrum between those two extremes.

We now set out to make the idea more concrete. Let us begin with the concept of region systems.

5.3 Region systems

A *region* is simply a subset of states of the world. A *region system* is a collection of regions such that no region is a subset of other regions in the collection and the union of all regions equals the set of all possible states of the world. We shall use R to denote a region and $\mathcal{R}$ to denote a region system.

Region systems are to be used to restrict the regions that the oracle can choose to report.

There are many possible ways to construct a region system. A natural way is to create a region for each state by including its "nearby" states. Let us make this more precise. Each action has an intended effect. The intended effect of `move-forward`, for instance, is to move one step forward. We say a state s is *reachable in one step* from another state s' if there is an action whose intended effect is, when the world is currently in state s', to take the world into state s. A state s_k is *reachable in k steps* from another state s_0 if there are state s_1, ..., s_{k-1} such that s_{i+1} is reachable from s_i in one step for all $0 \leq i \leq k-1$. Any state is reachable from itself in 0 step.

For any non-negative integer k, the *radius-k region* for a state s is the set of states that are reachable from s in k or less steps. A *radius-k region system* is the one obtained by creating a radius-k region for each state and then removing, one after another, regions that are subsets of others.

When k is 0, the radius-k region system consists of singleton regions. On the other hand, if the k is large enough so that any state is reachable from any other state in k or less steps, there is only one region in the radius-k region system, which the set of all possible states.

For any non-negative s-function $f(s)$ and any region R, we call the quantity $supp(f, R) = \sum_{s \in R} f(s) / \sum_{s \in \mathcal{S}} f(s)$ the *degree of support* of f by R. If R supports f to degree 1, we say that R *fully supports* f.

5.4 Region observable POMDPs

Given a region system $\mathcal{R}$ and a POMDP $\mathcal{M}$, we construct a region observable POMDP $\mathcal{M}'$ by assuming that at each time point the agent not only obtains an observation by himself but also receives a report from an oracle who knows the true state of the world. The oracle does not report the true state itself. Rather he chooses from $\mathcal{R}$ one region that contains the true state and reports that region.

The amount of extra information provided by the oracle depends not only on the region system used but also on the way the oracle chooses regions. We suggest the following rule. Let $s_.$ be the previous true state of the world, $a_.$ be the previous action, and o be the current observation. The oracle should choose, among all the regions in $\mathcal{R}$ that contain the true state of the world, one that supports the function $P(s, o|s_., a_.)$ of s to the maximum degree. Where there is more than one such regions, choose the one that comes first in a predetermined ordering among the regions.

Here are the intuitions. If the previous world state $s_.$ were known to the agent, then his current belief state $b(s)$ would be proportional to $P(s, o|s_., a_.)$. In this case, the rule minimizes extra information in the sense that it supports the current belief state to the maximum degree. Also if the current observation is informative enough, being a landmark for instance, to ensure that the world state is in a certain region, then region chosen using the rule fully supports the current belief state. In such a case, no extra information is provided.

The probability $P(R|s, o, s_., a_.)$ of a region R being chosen under the above scheme is given by

$$P(R|s, o, s_., a_.) = \begin{cases} 1 & \text{if } R \text{ is the first region s.t. } s \in R \text{ and for any other region } R' \\ & \sum_{s' \in R} P(s', o|s_., a_.) \geq \sum_{s' \in R'} P(s', o|s_., a_.) \\ 0 & \text{otherwise.} \end{cases}$$

The region observable POMDP $\mathcal{M}'$ differs from the original POMDP $\mathcal{M}$ only in terms of observation; in addition to the observation o made by himself, the agent also receives a report R from the oracle. We shall denote an observation in $\mathcal{M}'$ by z and write $z=(o, R)$. Observation model of $\mathcal{M}'$ is given by

$$P(z|s, a_., s_.) = P(o, R|s, a_., s_.) = P(o|s, a_.)P(R|s, o, s_., a_.).$$

6 Solving region observable POMDPs

For any region R, let $\mathcal{B}_R$ be the set of belief states that are fully supported by R. For any region system $\mathcal{R}$, let $\mathcal{B}_\mathcal{R} = \cup_{R \in \mathcal{R}} \mathcal{B}_R$.

Given the region system $\mathcal{R}$, a set $\mathcal{V}'$ of s-functions is an *$\mathcal{R}$-covering* of another set $\mathcal{V}$ of s-functions if their induced b-functions are the same over $\mathcal{B}_\mathcal{R}$, i.e. if

$$\mathcal{V}'(b) = \mathcal{V}(b) \text{ for all } b \in \mathcal{B}_\mathcal{R}.$$

An $\mathcal{R}$-covering of $\mathcal{V}$ is *parsimonious* if none of its proper subsets are $\mathcal{R}$-coverings of $\mathcal{V}$.

6.1 Restricted value iteration

Let $\mathcal{R}$ be the region system used in the definition of the region observable POMDP $\mathcal{M}'$. It is easy to see that no matter what the current belief state b is, the next belief state b_+ must be in $\mathcal{B}_\mathcal{R}$. We assume that in $\mathcal{M}'$ the initial belief state is in $\mathcal{B}_\mathcal{R}$. Then all possible belief states the agent might have are in $\mathcal{B}_\mathcal{R}$. This implies that policies for $\mathcal{M}'$ need only be defined over $\mathcal{B}_\mathcal{R}$ and value iteration for $\mathcal{M}'$ can restricted to the subset $\mathcal{B}_\mathcal{R}$ of $\mathcal{B}$.

We shall restrict value iteration for $\mathcal{M}'$ to $\mathcal{B}_\mathcal{R}$ for the sake of efficiency. Doing so implies that the t-step optimal value function $V_t^{'*}$ of $\mathcal{M}'$ is defined only over $\mathcal{B}_\mathcal{R}$ and the Bellman residual is now $max_{b\in\mathcal{B}_\mathcal{R}}|V_t^{'*}(b) - V_{t-1}^{'*}(b)|$. To avoid confusion, we shall call it the *restricted Bellman residual*.

Since $V_t^{'*}$ is defined only over $\mathcal{B}_\mathcal{R}$, it is represented by a parsimonious $\mathcal{R}$-covering $\mathcal{V}_t^\mathcal{R}$ of $\mathcal{V}_t$ in the sense that for any $b\in\mathcal{B}_\mathcal{R}$,

$$V_t^{'*}(b) = \mathcal{V}_t(b) = \mathcal{V}_t^\mathcal{R}(b).$$

Consequently, we can carry out value iteration for $\mathcal{M}'$ implicitly by inductively computing parsimonious $\mathcal{R}$-coverings, instead of parsimonious coverings, of the $\mathcal{V}_t$'s. Parsimonious $\mathcal{R}$-coverings of $\mathcal{V}_t$ usually contains much less s-functions than parsimonious coverings of $\mathcal{V}_t$.

6.2 Regional coverings

Given a region R, a set $\mathcal{V}'$ of s-functions is an *R-covering* of another set $\mathcal{V}$ of s-functions if

$$\mathcal{V}'(b) = \mathcal{V}(b) \text{ for all } b \in \mathcal{B}_R, \text{ and}$$
$$\mathcal{V}'(b) \leq \mathcal{V}(b) \text{ for all } b \notin \mathcal{B}_R.$$

We shall sometimes refer to an R-covering simply as a *regional covering* where there is no need to refer to the region. An R-covering of $\mathcal{V}$ is *parsimonious* if none of its proper subsets are R-coverings of $\mathcal{V}$.

If for each region R in a region system $\mathcal{R}$, $\mathcal{V}_t^R$ is an R-covering of $\mathcal{V}_t$, then the union $\cup_{R\in\mathcal{R}}\mathcal{V}_t^R$ is $\mathcal{R}$-covering of $\mathcal{V}_t$. Parsimonious R-coverings are easier to compute than parsimonious $\mathcal{R}$-coverings because they are "local". We propose to carry out implicit value iteration for the region observable POMDP $\mathcal{M}'$ by inductively computing R-coverings of $\mathcal{V}_t$'s. To be more specific, let `updating` be a procedure that computes a parsimonious R-covering of $\mathcal{V}_t$ from parsimonious regional coverings of $\mathcal{V}_{t-1}$ and let `stop` be a procedure that determines, from parsimonious regional coverings of $\mathcal{V}_{t-1}$ and $\mathcal{V}_t$, whether the restricted Bellman residual has fallen below a predetermined threshold. Value iteration for $\mathcal{M}'$ can be implicitly carried out as follows.

Procedure `solvePomdp`($\mathcal{M}', \epsilon$)

- Input: $\mathcal{M}'$ — A region observable POMDP,
  ϵ — A positive number.
- Output: A set of s-functions.

1. $t = 0$.
2. (Let $\mathcal{R}$ be the region system used in the definition of $\mathcal{M}'$.) **For** $R \in \mathcal{R}$, let $\mathcal{V}_0^R$ consist of the only s-function 0.
3. **Do**
   - $t{=}t{+}1$.
   - **For** each $R \in \mathcal{R}$,

     $$\mathcal{V}_t^R = \texttt{updating}(R, \{\mathcal{V}_{t-1}^R | R \in \mathcal{R}\}).$$

   while `stop`($\{\mathcal{V}_t^R | R \in \mathcal{R}\}, \{\mathcal{V}_{t-1}^R | R \in \mathcal{R}\}, \epsilon$) = `no`.
4. Return $\cup_R \mathcal{V}_{t-1}^R$.

The next two sections show how to implement the procedures `updating` and `stop`.

7 Dynamic-programming updates

Previous algorithms, such as the witness algorithm (Cassandra *et al* 1994), for computing a parsimonious covering of $\mathcal{V}_t$ from a parsimonious covering of $\mathcal{V}_{t-1}$ can be adapted to compute parsimonious regional coverings of $\mathcal{V}_t$ from parsimonious regional coverings of $\mathcal{V}_{t-1}$. This section develops a simpler algorithm called *incremental pruning*. We have empirically shown elsewhere that incremental pruning is more efficient than the witness algorithm (Cassandra *et al* 1997).

7.1 Operations on sets of s-functions

Suppose $\mathcal{V}_1$ and $\mathcal{V}_2$ are two sets of s-functions. The *cross sum* $\mathcal{V}_1 + \mathcal{V}_2$ of $\mathcal{V}_1$ and $\mathcal{V}_2$ is defined to be the following set of s-functions:

$$\{V_1 + V_2 | V_1 \in \mathcal{V}_1, V_2 \in \mathcal{V}_2\}.$$

It is evident that the cross sum operation is commutative and associative. Hence one can talk about the cross sum of more than two sets of s-functions.

The *product* $\gamma\mathcal{V}$ of of a constant γ and and a set of s-functions $\mathcal{V}$ is defined by

$$\gamma\mathcal{V} = \{\gamma V | V \in \mathcal{V}\}.$$

For any function $f(s_+, s)$, the *matrix product* $\mathcal{V} * f$ of $\mathcal{V}$ and f is defined by

$$\mathcal{V} * f = \{\sum_{s_+} V(s_+) f(s_+, s) | V \in \mathcal{V}\}.$$

7.2 Relationship between $\mathcal{V}_t$ and $\mathcal{V}_{t-1}$

For any action a and any observation z_+, regard $P(s_+, z_+|s, a)$ as a function of s_+ and s. Define

$$\mathcal{V}_{a,z_+,t} = \gamma(\mathcal{V}_{t-1} * P(s_+, z_+|s, a)). \tag{11}$$

Enumerate all possible values of z_+ as 0, 1, ..., N. Define

$$\mathcal{V}_{a,t} = \{r(s, a)\} + \mathcal{V}_{a,0,t} + \mathcal{V}_{a,1,t} + \ldots + \mathcal{V}_{a,N,t}.$$

One can easily see from equation (10) that this set consists of state value functions of all those t-step policy trees p_t whose first actions are a. Consequently,

$$\mathcal{V}_t = \cup_a \mathcal{V}_{a,t}$$

7.3 Inductive computation of parsimonious regional coverings

Suppose a parsimonious R-covering $\mathcal{V}_{t-1}^R$ of $\mathcal{V}_{t-1}$ has been obtained for each region $R \in \mathcal{R}$. For any action a and observation $z_+ = (o_+, R_+)$, define

$$\hat{\mathcal{V}}_{a,z_+,t} = \gamma(\mathcal{V}_{t-1}^{R_+} * P(s_+, z_+|s, a)). \tag{12}$$

Also define

$$\begin{aligned} \hat{\mathcal{V}}_{a,t} &= \{r(s, a)\} + \hat{\mathcal{V}}_{a,0,t} + \hat{\mathcal{V}}_{a,1,t} + \ldots + \hat{\mathcal{V}}_{a,N,t}, \\ \hat{\mathcal{V}}_t &= \cup_a \hat{\mathcal{V}}_{a,t}. \end{aligned}$$

Lemma 1 *The set $\hat{\mathcal{V}}_{a,z_+,t}$ is a covering of $\mathcal{V}_{a,z_+,t}$. Hence it is an R-covering of $\mathcal{V}_{a,z_+,t}$ for any region R.*

Proof: For any belief state $b \in \mathcal{B}$,

$$\begin{aligned} \mathcal{V}_{a,z_+,t}(b) &= max_{V \in \mathcal{V}_{a,z_+,t}} \sum_s V(s) b(s) \\ &= \gamma \ max_{V \in \mathcal{V}_{t-1}} \sum_s [\sum_{s_+} V(s_+) P(s_+, z_+|s, a)] b(s) \\ &= \gamma k \ max_{V \in \mathcal{V}_{t-1}} \sum_{s_+} V(s_+) b_+(s_+), \end{aligned}$$

where $b_+(s_+) = \sum_s P(s_+, z_+|s, a) b(s)/k$ and $k = \sum_{s,s_+} P(s_+, z_+|s, a) b(s)$. If $z_+ = (o, R_+)$, then b_+ is a belief state in $\mathcal{B}_{\mathcal{R}_+}$. Since $\mathcal{V}_{t-1}^{R_+}$ is an R_+-covering of $\mathcal{V}_{t-1}$, we have

$$\begin{aligned} \mathcal{V}_{a,z_+,t}(b) &= \gamma k \ max_{V \in \mathcal{V}_{t-1}^{R_+}} \sum_{s_+} V(s_+) b_+(s_+) \\ &= \gamma \ max_{V \in \mathcal{V}_{t-1}^{R_+}} \sum_s [\sum_{s_+} V(s_+) P(s_+, z_+|s, a)] b(s) \\ &= \hat{\mathcal{V}}_{a,z_+,t}(b). \square \end{aligned}$$

Lemma 2 *Let R be a region. If $\mathcal{V}_1'$ and $\mathcal{V}_2'$ are R-coverings of $\mathcal{V}_1$ and $\mathcal{V}_2$ respectively, then*

1. *$\mathcal{V}_1'+\mathcal{V}_2'$ is an R-covering of $\mathcal{V}_1+\mathcal{V}_2$.*
2. *$\mathcal{V}_1'\cup\mathcal{V}_2'$ is an R-covering of $\mathcal{V}_1\cup\mathcal{V}_2$.* □

Lemma 3 *Let R be a region. If $\mathcal{V}_1$ is a parsimonious R-covering of $\mathcal{V}_2$ and $\mathcal{V}_2$ is an R-covering of $\mathcal{V}_3$, then $\mathcal{V}_1$ is a parsimonious R-covering of $\mathcal{V}_3$.* □

Those three lemmas tell us that $\hat{\mathcal{V}}_{a,t}$ is an R-covering of $\mathcal{V}_{a,t}$ and $\hat{\mathcal{V}}_t$ is an R-covering of $\mathcal{V}_t$. Moreover, a parsimonious R-covering of $\hat{\mathcal{V}}_t$ is also a parsimonious R-covering of $\mathcal{V}_t$.

Let $\texttt{purge}(\mathcal{V}, R)$ be a procedure that returns a parsimonious R-covering of $\mathcal{V}$. One implementation of the procedure will be given in Appendix B. An R-covering of $\hat{\mathcal{V}}_t$ can be found by

$$\texttt{purge}(\cup_a \texttt{purge}(\hat{\mathcal{V}}_{a,t}, R), R).$$

Both the witness algorithm and incremental pruning make use of this fact.

7.4 Incremental pruning

Incremental pruning and the witness algorithm differ in their ways of computing $\texttt{purge}(\hat{\mathcal{V}}_{a,t}, R)$. Incremental pruning computes it by interleaving purging and cross sum as follows:

$$\texttt{purge}(\ldots(\texttt{purge}(\texttt{purge}(\{r(s,a)\} + \hat{\mathcal{V}}_{a,0,t} + \hat{\mathcal{V}}_{a,1,t}, R) + \hat{\mathcal{V}}_{a,2,t}, R) + \ldots + \hat{\mathcal{V}}_{a,N,t}, R).$$

The correctness of incremental pruning is guaranteed by lemmas 2 and 3.

An efficiency consideration. For any region R and any action a, define set

$$\mathcal{Z}_{R,a} = \{z_+ | max_{s\in R, s_+} P(s_+, z_+ | s, a) > 0\}.$$

It is the set of possible next observations given that the current true state of the world is in R and that the current action is a. Let R_a be the set of world states that one can possibly reach from some states in R by performing action a, i.e. $R_a = \{s_+ | max_{s\in R} P(s_+ | s, a) > 0\}$. An observation $z_+ = (o_+, R_+)$ is outside $\mathcal{Z}_{R,a}$ if R_+ and R_a do not intersect.

Lemma 4 *Enumerate all the elements of the set $\mathcal{Z}_{R,a}$ as $0, 1, \ldots, M$. Then*

$$\{r(s,a)\} + \hat{\mathcal{V}}_{a,0,t} + \hat{\mathcal{V}}_{a,1,t} + \ldots + \hat{\mathcal{V}}_{a,M,t}$$

is an R-covering of $\hat{\mathcal{V}}_{a,t}$.

Proof: Let z_+ be outside the set $\mathcal{Z}_{R,a}$. Then $P(s_+, z_+ | s, a) = 0$ for all $s \in R$. Hence for any $V \in \hat{\mathcal{V}}_{a,z_+,t}$, $V(s) = 0$ for all $s \in R$. The lemma follows. □

According to this lemma, a parsimonious R-covering of $\hat{\mathcal{V}}_{a,t}$ can be computed by

$$\texttt{purge}(\ldots(\texttt{purge}(\texttt{purge}(\{r(s,a)\} + \hat{\mathcal{V}}_{a,0,t} + \hat{\mathcal{V}}_{a,1,t}, R) + \hat{\mathcal{V}}_{a,2,t}, R) + \ldots + \hat{\mathcal{V}}_{a,M,t}, R).$$

7.5 An algorithm

The forgoing discussions lead to the following procedure for computing parsimonious regional coverings of $\mathcal{V}_t$ from parsimonious regional coverings of $\mathcal{V}_{t-1}$.

Procedure `updating`$(R, \{\mathcal{V}_{t-1}^{R'} | R' \in \mathcal{R}\})$:

- Inputs: R — A region. And for any region R',
  $\mathcal{V}_{t-1}^{R'}$ — A parsimonious R'-covering of $\mathcal{V}_{t-1}$.
- Output: A parsimonious R-covering of $\mathcal{V}_t$.

1. **For** each action a,
$$\mathcal{V}_{a,t}^{R} = \texttt{incremental-pruning}(R, a, \{\mathcal{V}_{t-1}^{R'} | R' \in \mathcal{R}\}).$$
2. Return $\texttt{purge}(\cup_a \mathcal{V}_{a,t}^{R}, R)$.

Subprocedure `incremental-pruning`$(R, a, \{\mathcal{V}_{t-1}^{R'} | R' \in \mathcal{R}\})$:

1. Compute the set $\mathcal{Z}_{R,a}$ and enumerate its members as $0, 1, \ldots, M$.
2. For i=0 to M, compute the set $\hat{\mathcal{V}}_{a,i,t}$ from $\{\mathcal{V}_{t-1}^{R'} | R' \in \mathcal{R}\}$ and
$$\hat{\mathcal{V}}_i = \texttt{purge}(\hat{\mathcal{V}}_{a,i,t}, R).$$
3. Let $\mathcal{V}$=$\{r(s,a)\} + \hat{\mathcal{V}}_0$.
4. **For** i=1 to M,
   - $\mathcal{V}$=$\mathcal{V}$+$\hat{\mathcal{V}}_i$.
   - $\mathcal{V}$=`purge`($\mathcal{V}$, R). **Endfor**
5. Return $\mathcal{V}$.

8 The stopping condition

This section shows how to determine if the restricted Bellman residual falls below a predetermined threshold ϵ from regional coverings $\mathcal{V}_t^R$ of $\mathcal{V}_t$ and regional coverings $\mathcal{V}_{t-1}^R$ of $\mathcal{V}_{t-1}$.

Proposition 1 *In a POMDP if the reward function $r(s,a)$ is non-negative, then the t-step optimal value function $V_t^*(b)$ in non-decreasing with t.*

Proof: For any belief state b, let p_{t-1} be a $t-1$-step policy tree such that $V_{p_{t-1}}(b) = V_{t-1}^*(b)$. Grow p_{t-1} at the end by attaching an arbitrary o-rooted 1-step policy tree to each leaf node of p_{t-1}. What results in is a t-step policy tree. Denote it by p_t. Since the reward function is non-negative, $V_{p_t}(b) \geq V_{p_{t-1}}(b)$. Hence

$$V_t^*(b) \geq V_{p_t}(b) \geq V_{p_{t-1}}(b) = V_{t-1}^*(b).$$

The proposition is proved. □

As explained in Section 3.3, one can assume always that the reward function is non-negative without losing any generality. This assumption enables us to make use of the above proposition and rewrite the restricted Bellman residual as

$$max_{b \in \mathcal{B}_{\mathcal{R}}}(V_t^*(b) - V_{t-1}^*(b)).$$

For any two s-functions V and V', we say V *ϵ-dominates* V' at a belief state b if $V(b) \geq V'(b) + \epsilon$. Let $\mathcal{V}$ be a set of s-functions. V *ϵ-dominates* $\mathcal{V}$ at a belief state b is if either $\mathcal{V}$ is empty or V ϵ-dominates each member of $\mathcal{V}$ at b. The following lemmas is obvious.

Lemma 5 *The restricted Bellman residual is larger than ϵ if and only if there exists a region R and an s-function $V \in \mathcal{V}_t^R$ such that V ϵ-dominates $\mathcal{V}_{t-1}^R$ at some belief state in $\mathcal{B}_R$.* □

Let `dominate`$(V, \mathcal{V}, R, \epsilon)$ be a procedure that returns a belief state in $\mathcal{B}_R$ at which V ϵ-dominates $\mathcal{V}$ if one exists and `nil` if not. One implementation of this procedure can be found in Appendix B. The following procedure returns `yes` is the restricted Bellman residual falls below ϵ and `no` otherwise.

Procedure `stop`$(\{\mathcal{V}_t^R | R \in \mathcal{R}\}, \{\mathcal{V}_{t-1}^R | R \in \mathcal{R}\}, \epsilon)$

- Inputs: ϵ — A positive number, and for any region R
  $\mathcal{V}_t^R$ — An R-covering of $\mathcal{V}_t$,
  $\mathcal{V}_{t-1}^R$ — An R-covering of $\mathcal{V}_{t-1}$.
- Outputs: `yes` — If the restricted Bellman residual $\leq \epsilon$,
  `no` — Otherwise.

1. **For1** each region R,
   For2 each $V \in \mathcal{V}_t^R$,
   Return `no` if `dominate`$(V, \mathcal{V}_{t-1}^R, R, \epsilon) \neq$ `nil`.
   Endfor2
   EndFor1
2. Return `yes`.

9 Making decisions

Solving the region observable POMDP $\mathcal{M}'$ gives us a list of s-functions, which will be henceforth denoted by $\mathcal{V}'_{t-1}$. It represents the $t-1$-step value function $V'^*_{t-1}(b)$ of $\mathcal{M}'$ in the sense that $V'^*_{t-1}(b) = \mathcal{V}^R_{t-1}(b)$ for any $b \in \mathcal{B}_{\mathcal{R}}$ and hence enables us to compute the greedy policy π' for $\mathcal{M}'$ based on V'^*_{t-1}.

A policy π for the original POMDP $\mathcal{M}$ can be obtained through either value approximation or policy approximation. *Value approximation* means to approximate the $t-1$-step optimal value function $V^*_{t-1}(b)$ of $\mathcal{M}$ by using $\mathcal{V}'_{t-1}(b)$ and use the greedy policy based on $\mathcal{V}'_{t-1}(b)$. Under this scheme, the action for any belief state b is given by

$$\pi(b) = arg\ \ max_a[r(b,a) + \gamma \sum_{o_+} P(o_+|b,a)\mathcal{V}'_{t-1}(b_+)],$$

where b_+ is a shorthand for $b_+(.|o_+, b, a)$.

Policy approximation means to directly use the policy π' for $\mathcal{M}'$ as a policy for $\mathcal{M}$. One issue here is that π' is defined only over $\mathcal{B}_\mathcal{R}$. For each belief state b in $\mathcal{B}_\mathcal{R}$, we can set $\pi(b)=\pi'(b)$. What about belief states outside $\mathcal{B}_\mathcal{R}$? A natural solution is as follows. For any belief state $b \notin \mathcal{B}_\mathcal{R}$, *project* it onto $\mathcal{R}$, i.e. find a region R in $\mathcal{R}$ that supports b to the largest degree and restrict b onto R to get another belief state b' as follows:

$$b'(s) = \begin{cases} \frac{b(s)}{\sum_{s \in R} b(s)} & \text{if } s \in R \\ 0 & \text{otherwise.} \end{cases}$$

Then set $\pi(b)=\pi'(b')$.

9.1 Information gathering

Simulation experiments have shown that the two approximate policies given above are close to optimal when the agent has a good idea about the true state of the world at all times (see Subsection 10.6). However, they can be arbitrarily bad when the agent is confused about the true state of the world. In general, they should be used together with some information gathering scheme which takes over when the agent is confused about the true state of the world.

Two questions arise. When information gathering should take over? How information gathering should be carried out?

Define *the degree of support of a belief state b by a region system $\mathcal{R}$* to be $supp_\mathcal{R}(b)= max_{R \in \mathcal{R}} supp_R(b)$. In our experiments, information gathering took over when the degree of support of the current belief state by the radius-1 region system fall below a certain threshold, which was set at 0.8.

For any belief state b, its *entropy* is given by $EN(b)= -\sum_s b(s) log(b(s))$. The larger the entropy, the more uncertain the agent is about the true state of the world. Let b be the current belief state. If action a is executed, the the expected entropy of the next belief state is given by

$$EN(b,a) = \sum_{o_+} EN(b_+)P(o_+|b,a),$$

where b_+ is a shorthand for $b_+(.|o_+, b, a)$. One approach to information gathering is to execute the action that minimizes the expected entropy of the next belief state (Cassandra *et al* 1996).

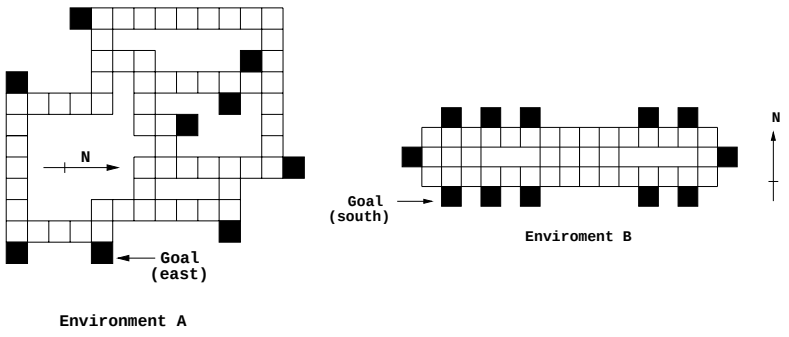

Figure 2: Synthetic Office Environments.

10 Simulation experiments

Simulation experiments have been carried out with synthetic office environments borrowed from Cassandra *et al* (1996). POMDPs for those environments were previously approximated by using MDPs (Cassandra *et al* 1996). We transformed the POMDPs into region observable POMDPs by using the radius-1 region systems and solved the region observable POMDPs. We found that control policies produced by the region observable POMDPs are significantly better than those produced the MDPs. This section describes the experiment setups and reports the comparisons.

10.1 Synthetic office environments

Two synthetic office environments were used in our experiments. Their layouts are shown in Figure 2, where squares represent locations. Each location is represented as four states in the POMDP model, one for each orientation. The dark locations are rooms connected to corridors by doorways.

In each environment, a robot needs to reach the goal location with the correct orientation. At each time point, the robot can execute one of the following actions: `move-forward`, `turn-left`, `turn-right`, and `declare-goal`. The two sets of action models given in the following table were used.

Action	Standard outcomes	Noisy outcomes
`move-forward`	N (0.11), F (0.88), F-F (0.01)	N (0.2), F (0.7), F-F (0.1)
`turn-left`	N (0.05), L (0.9), L-L (0.05)	N (0.15), L (0.7), L-L (0.15)
`turn-right`	N (0.05), R (0.9), R-R (0.05)	N (0.15), R (0.7), R-R (0.15)
`declare-goal`	N (1.0)	N (1.0)

For the action `move-forward`, the term F-F (0.01) means than with probability 0.01 the robot actually moves two steps forward. The other terms are to be interpreted similarly. If an outcome cannot occur in a certain state of the world, then the robot is left in the last state before the impossible outcome.

In each state, the robot is able to perceive in each of three nominal directions (front,

left, and right) whether there is a `doorway`, `wall`, `open`, or it is `undetermined`. The following two sets of observation models were used:

Actual case	Standard observations	Noisy observations
`wall`	`wall` (0.90), `open` (0.04), `doorway` (0.04), `undetermined` (0.02)	`wall` (0.70), `open` (0.19), `doorway` (0.09), `undetermined` (0.02)
`open`	`wall` (0.02), `open` (0.90), `doorway` (0.06), `undetermined` (0.02)	`wall` (0.19), `open` (0.70), `doorway` (0.09), `undetermined` (0.02)
`doorway`	`wall` (0.15), `open` (0.15), `doorway` (0.69), `undetermined` (0.01)	`wall` (0.15), `open` (0.15), `doorway` (0.69), `undetermined` (0.01)

10.2 Approximations in solving region observable POMDPs

Approximations were made when solving the region observable POMDP $\mathcal{M}'$. Let R be a region and σ be a positive number. A set $\mathcal{V}'$ of s-functions is an (R, σ)-*covering* of another set $\mathcal{V}$ of s-functions if

$$\mathcal{V}'(b) + \sigma \geq \mathcal{V}(b) \text{ for all } b \in \mathcal{B}_R, \text{ and}$$

$$\mathcal{V}'(b) \leq \mathcal{V}(b) \text{ for all } b \notin \mathcal{B}_R.$$

In all the algorithms, we replaced the procedure $\texttt{purge}(\mathcal{V}, R)$ with another procedure $\texttt{purge}(\mathcal{V}, R, \sigma)$ that returns an (R, σ)-covering of $\mathcal{V}$. And σ was set to be 0.001.

The POMDPs for both environments were solved by using a SUN SPARC20. The threshold for the restricted Bellman residual was set at 0.008. The POMDP for environment A took 5984 CPU seconds to solve, that the POMDP for environment B took 3952 seconds.

10.3 Initialization schemes

Three initialization schemes were used in the experiments. In the first scheme, a starting state was randomly generated and the robot was informed of state. In the second scheme, a pair of states was first randomly generated and the robot was informed of the pair. Then one of the two states was randomly picked to be the starting state and the robot was not informed of which. In the third scheme, a starting state was randomly generated and the robot was not informed of state.

For each scheme, 1000 trials were conducted and each trial was allowed to run a maximum number of 300 steps. Statistics were summarized by a function $g(n)$, which stands for the number of trials successfully completed in n or less steps.

10.4 Omniscient agents

Statistics were also collected for an *omniscient* agent who knows the true state of the world at all times. This enables us to tell that a policy is close to optimal in some cases. More specifically, the performance of any policy can be no better than that of an optimal policy, which in turn can be no better than the performance of the omniscient agent. If the performance of an policy is close to that of the omniscient agent, then it must be close

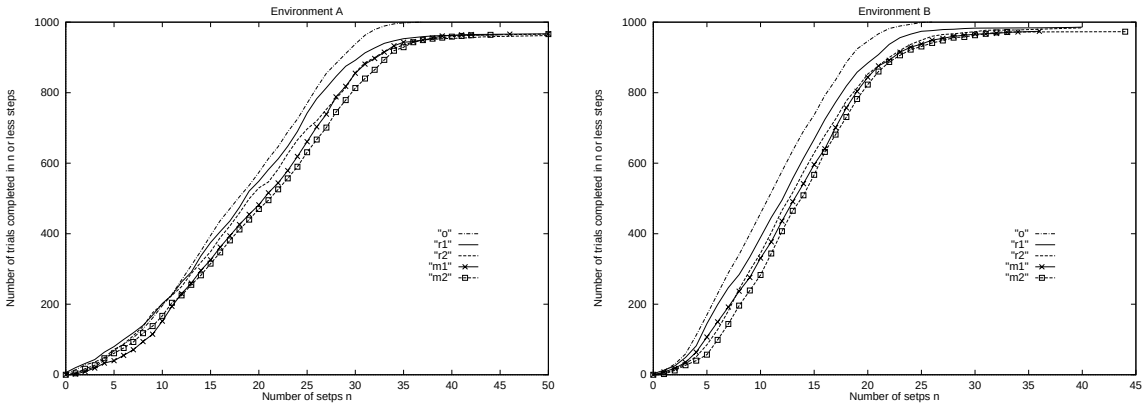

Figure 3: Statistics for experiment 1.

to optimal. It should be noted, however, that a policy might still be close optimal even when its performance is significantly worse that of the omniscient agent.

10.5 Value approximation vs policy approximation

Let $\mathcal{M}$ be a POMDP and let $\mathcal{M}^k$ be the region observable POMDP obtained from $\mathcal{M}$ by using the radius-k region system. As pointed out in Section 9, a policy for $\mathcal{M}$ can be obtained from the solution of $\mathcal{M}^k$ through either value approximation or policy approximation. Our experiments with the synthetic office environments have shown that, at least for k=0 or 1, the policy obtained through value approximation is never significantly better than the one obtained through policy approximation and is often significantly worse.

In the following, we shall only compare policies obtained through policy approximation. Let us introduce several notations. When k=0, $\mathcal{M}^k$ is an MDP. The policy for $\mathcal{M}$ obtained from the solution of $\mathcal{M}^0$ through policy approximation will be denoted by π^m. On the other hand, the policy obtained from the solution of $\mathcal{M}^1$ through policy approximation will be denoted by π^r. The g-function for the omniscient agent will be denoted by o. The g-functions for π^m and π^r under the three initialization schemes should be denoted by mi and ri $(i = 1, 2, 3)$ respectively.

10.6 Experiment 1

In the first experiment, standard action and observation models were used and trials were initialized by using the first two schemes. Statistics are summarized in Figure 3.

The success rates for both π^m and π^r are high and roughly the same in both environments. Among all the trials successfully completed, π^r took on average about 2 steps less than π^m in environment A and about 1.5 steps less in environment B. There is about 10% improvement in both environments. The improvement is significant considering the fact that the performance of π^r is close to that of the omniscient agent and hence close to optimal.

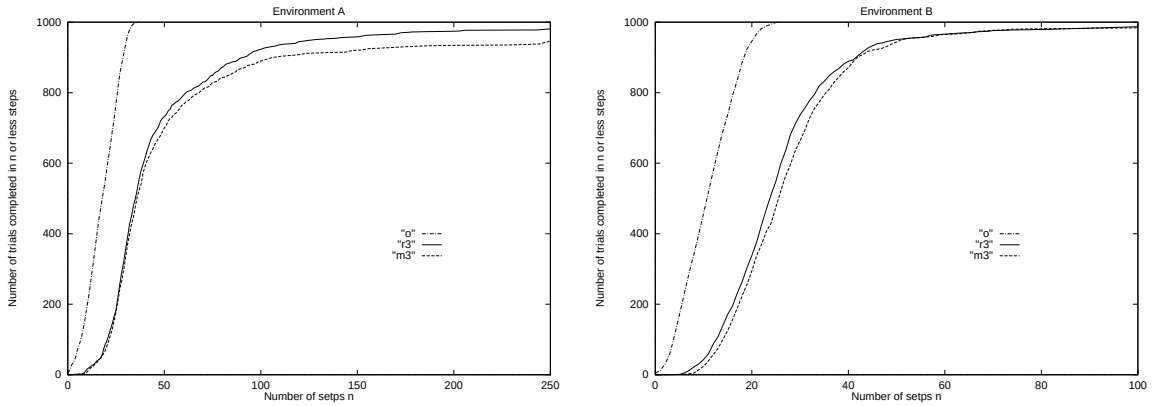

Figure 4: Statistics for experiment 2.

A couple of other facts are also worth mentioning. Firstly, both π^m and π^r performanced worse under the second initialization scheme than under the first. This is expected since the robot had less information about the starting state under the second scheme. Secondly, we have found that information gathering did not affect the performances of the two policies either way. This is because the robot had an good idea about the true state of the world at all times and information gathering was never really needed.

10.7 Experiment 2

In the second experiment, standard action and observation models were used and trials were initialized by using the third scheme. In this case, the robot was completely ignorant about the starting state. Hence the polices π^r and π^m were augmented with information gathering. The statistics are shown in Figure 4.

As one would expect, the robot took on average many more steps to achieve a goal than in the previous experiment, while the success rate of π^r remained roughly the same.

The success rate of π^r is higher than that of π^m in environment A and it is the same as that of π^m in environment B. In both environments, π^r took on average less steps to achieve a goal than π^m.

It is worthwhile to point out that both π^r and π^m performed considerably worse without information gathering.

10.8 Experiment 3

In the third experiment, noisy action and observation models were used. In this case, the robot could easily get confused about the state of world even under the first and the second initialization schemes. So, the policies π^r and π^m were augmented with information gathering for all cases. Statistics are shown in Figure 5. For clarity, data for environment A with the first initialization scheme and data for environment B with the second initialization scheme are not included.

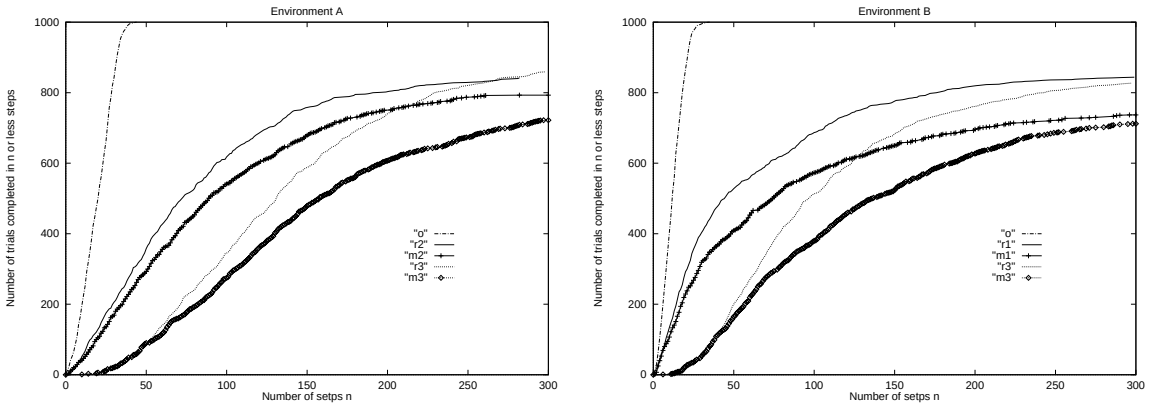

Figure 5: Statistics for experiment 3.

We see that π^r enjoyed a much higher success rate than π^m, especially under the third initialization scheme. It also took on average much less steps to achieve a goal than π^m, especially when the goal is "difficult".

Again both π^r and π^m performed considerably worse without information gathering.

11 Conclusions and future directions

An agent who acts in a partially observable stochastic domain typically does not know the true state of the world. However, he often has a good idea about it. This paper has proposed a way to exploit such problem characteristics in solving POMDPs approximately. The basic idea is to transform a POMDP into a region observable POMDP by assuming an oracle who reports that the true state is in a certain region. If the agent already knows that the state is roughly in a region and the oracle reports that region, then not much extra information is provided. In such a case, the region observable POMDP should be a good approximation of the original POMDP.

The region observable POMDP is easier to solve than the original POMDP due to the fact that the state of the world is always known to be in some region. We have developed an algorithm for solving region observable POMDPs, alongside a new algorithm for dynamic-programming updates.

POMDPs were previously approximated by using MDPs. It has been empirically shown that region observable POMDPs lead to significantly better policies than MDPs.

Even though much easier then general POMDPs, region observable POMDPs are still difficult to solve. The region observable POMDPs for the two synthetic office environments took about 4000 and 6000 seconds CPU time respectively. Speedup is necessary. We are currently exploring the following idea. For regions R far away from the goal, the parsimonious R-coverings of $\mathcal{V}_t$ remain unchanged when t is small and for regions close to the goal, the coverings remain unchanged when t becomes large. Preliminary experiments with MDPs have shown that this idea can speed up the solution process substantially.

As it stands, the method proposed in this paper computes policies without considering

the costs of information gathering and then uses the policies together with information gathering. Better policies can be obtained by taking the costs of information gathering into consideration when computing the policies.

Finally, the information gathering scheme described in Section 9.2 is basically a greedy approach. It recommends the action that would reveal the maximum amount of information for one step. A global approach would lead to better performances.

Acknowledgement

The paper has benefited from discussions with Anthony R. Cassandra and Michael Littman. Research was supported by Hong Kong Research Council under grants HKUST 658/95E and Hong Kong University of Science and Technology under grant DAG96/97.EG01(RI).

References

[1] D. P. Bertsekas (1987), *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall.

[2] C. Boutillier, R. Dearden and M. Goldszmidt (1995), Exploiting structures in policy construction, In *Proceedings of IJCAI'95*. pp. 1104-1111.

[3] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman (1994), Acting optimally in partially observable stochastic domains, AAAI Proc., July 31-August 4, Seattle, Washington, pp 1023-1028.

[4] A. R. Cassandra, L. P. Kaelbling, and J. Kurien (1996), Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation, TR CS-96-17, Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

[5] A. R. Cassandra, M. L. Littman, and N. L. Zhang (1997), Comparison of exact algorithms for partially observable Markov decision processes, to be submitted.

[6] H. T. Cheng (1988), Algorithms for partially observable Markov decision processes, PhD thesis, University of British Columbia, Vancouver, BC, Canada.

[7] T. L. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson (1993), Planning with deadlines in stochastic domains, In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC.

[8] T. L. Dean and M. P. Wellman (1991), *Planning and Control*, Morgan Kaufmann.

[9] J. N. Eagle (1984), The optimal search for a moving target when the search path is constrained, *Operations Research*, 32(5), pp. 1107-1115.

[10] M. L. Littman (1994), The witness algorithm: Solving partially observable Markov decision processes, TR CS-94-40, Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

[11] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling (1995), Efficient dynamic-programming updates in partially observable Markov decision processes, TR CS-95-19, Department of Computer Science, Brown University, Providence, Rhode Island 02912, USA.

[12] W. S. Lovejoy (1991a), A survey of algorithmic methods for solving partially observable Markov decision processes, *Annals of Operations Research*, 28 (1), pp. 47-65.

[13] W. S. Lovejoy (1991b), Computationally feasible bounds for partially observed Markov decision processes, *Operations Research*, 39 (1), pp. 162-175.

[14] G. E. Monahan (1982), A survey of partially observable Markov decision processes: theory, models, and algorithms, *Management Science*, 28 (1), pp. 1-16.

[15] R. Parr and S. Russell (1995), Approximating optimal polices for partially observable stochastic domains, In *Proceedings of IJCAI'95*, pp. 1088-1094.

[16] M. L. Puterman (1990), Markov decision processes, in D. P. Heyman and M. J. Sobel (eds.), *Handbooks in OR & MS.*, Vol. 2, pp. 331-434, Elsevier Science Publishers.

[17] E. J. Sondik (1971), The optimal control of partially observable Markov processes, PhD thesis, Stanford University, Stanford, California, USA.

[18] C. C. White III (1991), Partially observed Markov decision processes: A survey. *Annals of Operations Research*, 32.

[19] D. J. White (1993), *Markov Decision Processes*, John Wiley & Sons.

[20] C. C. White III and W. T. Scherer (1994), Finite-memory suboptimal design for partially observed Markov decision processes, *Operations Research*, 42(3), pp. 440-455.

Appendix A: Proof of Theorem 3

Lemma 6 *The b-function induced by a set $\mathcal{V}$ of s-functions is convex in the sense that for any two belief states b_1 and b_2, and any $0<\alpha<1$,*

$$\mathcal{V}(\alpha b_1+(1-\alpha)b_2) \leq \alpha\mathcal{V}(b_1) + (1-\alpha)\mathcal{V}(b_2).$$

Proof:

$$\begin{aligned}
\mathcal{V}(\alpha b_1+(1-\alpha)b_2) &= max_{V\in\mathcal{V}} V(\alpha b_1+(1-\alpha)b_2) \\
&= max_{V\in\mathcal{V}}[\alpha V(b_1) + (1-\alpha)V(b_2)] \\
&\leq \alpha\, max_{V\in\mathcal{V}} V(b_1) + (1-\alpha) max_{V\in\mathcal{V}} V(b_2) \\
&= \alpha\mathcal{V}(b_1) + (1-\alpha)\mathcal{V}(b_2).\Box
\end{aligned}$$

Proof of Theorem 3: Let $\mathcal{V}$ be a set of s-functions and let $\mathcal{V}^*$ and $\mathcal{U}^*$ be two different parsimonious representations of $\mathcal{V}$. It suffices to show that $\mathcal{U}^*$ cannot contain less members than $\mathcal{V}^*$.

Since $\mathcal{V}^*$ is a parsimonious representation of $\mathcal{V}$, for any $V\in\mathcal{V}^*$, there exists at least one belief state b such that $V(b)>V'(b)$ for any other $V'\in\mathcal{V}^*$. Call such a belief state a *witness point* of V. If b is a witness point of V, than $V(b)=\mathcal{V}(b)$. Different members of $\mathcal{V}^*$ have different witness points.

If the cardinality of $\mathcal{U}^*$ were smaller than that of $\mathcal{V}^*$, then there must be a $U\in\mathcal{U}^*$ and two s-functions $V_1, V_2\in\mathcal{V}^*$ such that $U(b_1)=V_1(b_1)$ and $U(b_2)=V_2(b_2)$, where b_i is a witness point V_i. For any $0<\alpha<1$,

$$\begin{aligned}
&\mathcal{V}(\alpha b_1+(1-\alpha)b_2) \geq U(\alpha b_1+(1-\alpha)b_2) = \alpha U(b_1)+(1-\alpha)U(b_2) \\
&= \alpha V_1(b_1)+(1-\alpha)V_2(b_2) = \alpha\mathcal{V}(b_1)+(1-\alpha)\mathcal{V}(b_2).
\end{aligned}$$

By the convexity of $\mathcal{V}$, it must be the case that

$$\mathcal{V}(\alpha b_1+(1-\alpha)b_2) = \alpha\mathcal{V}(b_1)+(1-\alpha)\mathcal{V}(b_2) = \alpha V_1(b_1)+(1-\alpha)V_2(b_2).$$

Since b_1 is a witness point of V_1, when $0<\alpha<1$ is close to 1 enough, $V_1(\alpha b_1+(1-\alpha)b_2) > V'(\alpha b_1+(1-\alpha)b_2)$ for any other $V'\in\mathcal{V}^*$. For such an α

$$V_1(\alpha b_1+(1-\alpha)b_2) = \mathcal{V}(\alpha b_1+(1-\alpha)b_2) = \alpha V_1(b_1)+(1-\alpha)V_2(b_2).$$

This implies that $V_1(b_2)=V_2(b_2)$, contradicting the fact that b_2 is a witness point of V_2. The theorem is proved. $\Box$

Appendix B: Domination and Purging

This appendix describes implementations of the procedures `dominate`$(V, \mathcal{V}, R, \epsilon)$ and `purge`(V, R). They were not given in the main text because they are minor adaptations of existing algorithms.

The procedure `dominate`$(V, \mathcal{V}, R, \epsilon)$ is supposed to returns a belief state in $\mathcal{B}_R$ at which the s-function V ϵ-dominates the set of s-functions $\mathcal{V}$. If such a belief state does not exist, it is supposed to return `nil`. It can be implemented as follows.

Procedure `dominate`$(V, \mathcal{V}, R, \epsilon)$

- Inputs: V — An s-function, $\mathcal{V}$ — A set of s-functions,
  R — A region, ϵ — A nonnegative number.
- Output: A belief state in $\mathcal{B}_R$ or `nil`.

1. **If** $\mathcal{V}=\emptyset$, return an arbitrary belief state in $\mathcal{B}_R$.
2. Solve the following linear program:

   Variables: x, $b(s)$ for each $s \in R$.
   Maximize: x
   Constraints:

$$\sum_{s \in R} V(s)b(s) \geq x + \sum_{s \in R} V'(s)b(s) \text{ for all } V' \in \mathcal{V},$$
$$\sum_{s \in R} b(s) = 1$$
$$b(s) \geq 0 \text{ for all } s \in R.$$

3. **If** $x \geq \epsilon$, return b, **else** return `nil`.

The procedure `purge`$(\mathcal{V}, R)$ is supposed to return a parsimonious R-covering of a set of s-functions $\mathcal{V}$. To implement it, we need two subroutines.

An s-function $V(s)$ *pointwise dominates* another s-function $V'(s)$ in a region R if $V(s) \geq V'(s)$ for all $s \in R$. The subroutine `pointwisePurge`$(\mathcal{V}, R)$ returns a minimal subset $\mathcal{V}'$ of $\mathcal{V}$ such that each s-function in $\mathcal{V}$ is pointwise dominated in the region R by at least one s-function in $\mathcal{V}'$. Implementation of this subroutine is straightforward.

The subroutine `best`$(b, \mathcal{V})$ returns an s-function in $\mathcal{V}$ that dominates all other s-functions in $\mathcal{V}$ at belief state b. Implementation of the subroutine is straightforward except for the issue of tie breaking. Arbitrary tie breaking can lead `purge`$(\mathcal{V}, R)$ to yield an R-representation of $\mathcal{V}$ that is not parsimonious. A correct way to break ties is as follows. Fix an ordering among states in R. This induces a lexicographic ordering among all s-functions. Among the tied s-functions, chose the one that is largest under the lexicographic ordering (Littman 1994).

The following implementation of `purge` is based on Lark's algorithm (White 1991).

Procedure `purge`$(\mathcal{V}, R)$

- Inputs: $\mathcal{V}$ — A set of s-functions,
  R — A region.
- Output: A parsimonious R-covering of $\mathcal{V}$.

1. $\mathcal{V}$ = `pointwisePurge`$(\mathcal{V}, R)$.
2. $\mathcal{V}^R = \emptyset$.
3. **While** $\mathcal{V} \neq \emptyset$,
   - Pick an s-function V from $\mathcal{V}$.
   - b=`dominate`$(V, \mathcal{V}^R, R, 0)$.
   - **If** b =`nil`, remove V from $\mathcal{V}$.
   - **Else** remove `best`$(b, \mathcal{V})$ from $\mathcal{V}$ and add it to $\mathcal{V}^R$.

   Endwhile
4. Return $\mathcal{V}^R$.